I Want to Ride My BIKE: Customizing Bit-Flipping Decoders for Ephemeral Key KEMs

Davide De Zuane[1,2], Marco Gianvecchio[3], Alessandro Annechini[3], Paolo Santini[1], Marco Baldi[1], Alessandro Barenghi[3], and Gerardo Pelosi[3]

[1] Marche Polytechnic University, Ancona, Italy
[2] IMT School for Advanced Studies, Lucca, Italy
[3] Politecnico di Milano, Milano, Italy

Abstract. Large public keys and ciphertexts remain a major obstacle to the deployment of post-quantum cryptosystems in applications currently relying on compact classical schemes. In code-based post-quantum key encapsulation mechanisms, reducing the Decoding Failure Rate (DFR) generally requires larger parameters, further increasing communication and storage overheads. Ephemeral key-pair scenarios allow a different trade-off: ideally, each secret key is used for a single decapsulation, limiting the opportunities to observe and exploit decryption failures.

We investigate how this setting can be used to reduce the public-key and ciphertext sizes of code-based key encapsulation mechanisms such as BIKE. We present the first optimized implementation of the BF-Max decoder: compared with conventional parallel bit-flipping decoders currently used by BIKE, BF-Max provides better error-correction capability, but its computational structure makes an efficient implementation less straightforward. Combining BF-Max with other decoders from the literature, we introduce decoding strategies that offer different trade-offs among DFR, execution time, public-key size, and ciphertext size.

Experimental results show that the proposed design reduces both public-key and ciphertext sizes by 7.8% compared with LeanBIKE, a BIKE variant specifically designed for ephemeral-key use, while retaining similar execution time and DFR. For additional context, the selected parameter set yields a combined public-key and ciphertext size that is 22% smaller than that of BIKE and smaller by a factor of 2.79 than that of HQC. Note that BIKE and HQC, unlike LeanBIKE and our design, target the long-term-key setting.

As a concrete application, we show that the solutions we propose make it possible to incorporate BIKE into constrained protocols, such as Internet Key Exchange version 2 (IKEv2), in which no code-based KEM would otherwise fit.

Keywords: Post-Quantum Cryptography · Key Encapsulation Mechanism · QC-MDPC · Bit-Flipping Decoders.

1 Introduction

The theory of error-correcting codes provides a rich source of computationally hard problems for the construction of post-quantum cryptographic primitives. This is reflected in the large number of code-based schemes submitted to the NIST Post-Quantum Cryptography standardization process, among which HQC [15] was ultimately selected for standardization. Code-based Key Encapsulation Mechanisms (KEMs) generally have larger public keys and ciphertexts than their lattice-based counterparts and, therefore, their use in real-world applications can be challenging. In code-based

KEMs such as Classic McEliece [5], BIKE [2], and HQC, decapsulation boils down to correcting an error vector whose weight cannot be too small for security reasons. Failure to decode translates into a decryption error. To keep the failure probability sufficiently low, the underlying error-correcting code must be large enough. As a consequence, both public-key and ciphertext sizes tend to increase.

In long-term key usage scenarios, IND-CCA2 security is typically required. For KEMs, this translates into requiring a Decoding Failure Rate (DFR)[4] below $2^{-\lambda}$, where λ is the security parameter. In contrast, for ephemeral deployments, weaker security notions may be sufficient, potentially allowing less stringent DFR requirements. Among the main code-based KEM proposals, BIKE is a QC-MDPC code-based KEM considered during the NIST post-quantum cryptography standardization process that offers particularly compact public keys and ciphertexts, significantly smaller than those of HQC. The reason why BIKE, despite being a Round 4 candidate in the NIST standardization process, was not selected is largely tied to the IND-CCA2 requirement and the lack of a sufficiently convincing DFR estimation methodology. The KEM construction was nevertheless considered sound [1].

Ephemeral KEMs. Many secure communication systems rely on ephemeral key-establishment mechanisms to provide forward secrecy. When a KEM is used in this setting, the corresponding keypair is generated for a single protocol execution and discarded immediately afterwards. As a consequence, the adaptive decryption capabilities assumed by the standard IND-CCA2 security model are no longer available to an attacker, and weaker security notions can be sufficient [24]. This observation is particularly relevant in the post-quantum setting. The large public keys and ciphertexts of QC-MDPC code-based KEMs can make packet-size limitations and protocol fragmentation practical concerns. Under conventional IND-CCA2 requirements, such constraints may discourage their adoption in favour of more compact alternatives.

Recent works have explored weaker security notions tailored to ephemeral settings. Some show that IND-CPA security can be sufficient in specific deployment scenarios, while others introduce intermediate notions such as IND-1CCA [6, 12, 20, 24]. These results suggest that the lack of a proof of IND-CCA2 security should not automatically preclude the use of QC-MDPC code-based KEMs, and this perspective is particularly relevant for BIKE. If weaker security notions are acceptable for a given deployment, alternative decoder designs may become attractive, offering different trade-offs among decoding complexity, decoding-failure rate, and implementation cost.

Fragmentation Issues in Network Protocols. In the ephemeral deployment scenarios discussed above, the size of public keys and ciphertexts becomes a practical concern. This is particularly true in protocols where key-establishment messages are transported over UDP and must fit within the path Maximum Transmission Unit (MTU). Let us consider three representative deployments: IKEv2 [14], DTLS 1.3 [18], and QUIC [13]. IKEv2 and DTLS are widely deployed security protocols operating over UDP, and QUIC is the transport protocol underlying HTTP/3. In all these cases, handshake message sizes directly affect deployment feasibility and protocol efficiency. A KEM whose public key or ciphertext exceeds the available space cannot be used as a drop-in replacement. The protocol must either introduce additional packets, explicit fragmentation mechanisms, or extra round trips. All these solutions increase implementation complexity and handshake overhead.

The consequences depend on the protocol. In IKEv2 [14], no fragmentation mechanism is available during the `IKE_SA_INIT` exchange. Although IPv4 permits network-layer fragmentation, relying on it would expose the protocol to fragmentation-based denial-of-service attacks. IPv6 imposes

[4] Averaged over keys and messages.

Table 1. Best-case payload budget available for a single ephemeral KEM in IKEv2 [14], DTLS 1.3 [18], and QUIC [13], after reserving 32 bytes for the classical KEX payload in a hybrid setting (e.g., an X25519 public key).

	IKEv2	DTLS 1.3	QUIC
Available space (B)	1308	1327	1301

stricter constraints on packet fragmentation. An additional round trip can be introduced to mitigate these issues, but at the cost of increased latency. In other protocols, such as DTLS 1.3 [18] and QUIC [13], fragmentation is feasible. It nevertheless remains a performance concern, since additional packets must be transmitted and processed during every handshake. Table 1 reports the space available for the public keys and ciphertexts of an ephemeral post-quantum KEM in IKEv2, DTLS 1.3, QUIC, and Rosenpass, when using a hybrid configuration that uses 32 bytes for the classic payload.

BIKE and Bit Flipping Decoders. In BIKE, the secret key consists of a pair of sparse polynomials $h_0(x), h_1(x) \in \mathbb{F}_2[x]/(x^r - 1)$. Decoding is applied to the syndrome vector corresponding to the polynomial $s(x) = h_0(x)e_0(x) + h_1(x)e_1(x)$, where $e_0(x)$ and $e_1(x)$ are random sparse polynomials sampled during encapsulation. Using coding-theory terminology, $(h_0(x), h_1(x))$ defines the parity-check matrix of a quasi-cyclic moderate-density parity-check (QC-MDPC) code. The polynomial $s(x)$ is the syndrome associated with the error vector $(e_0(x), e_1(x))$. Decoding consists of estimating the error vector from $(h_0(x), h_1(x))$ and $s(x)$. In practice, this task is typically performed through a Bit Flipping (BF) decoder. The key observation is that the syndrome is the sum of a relatively small number of cyclic shifts of $h_0(x)$ and $h_1(x)$. More precisely, it can be seen as the sum of a few polynomials of the form $x^i h_0(x)$ and $x^j h_1(x)$, where the indices i and j are determined by the support of $(e_0(x), e_1(x))$. Because of sparsity, only a limited number of cancellations occur. Consequently, if $x^i h_0(x)$ contributes to the syndrome, many of its coefficients are likely to appear in $s(x)$ as well, therefore by correlating cyclic shifts of $h_0(x)$ and $h_1(x)$ with $s(x)$, one can derive an estimate of the error vector. Indeed, if a shift shares many nonzero positions with the syndrome, it is reasonable to assume that the corresponding coefficient of the error vector is equal to one. Once an error position has been estimated, the syndrome can be updated by adding the corresponding shifted polynomial. When the estimate is correct, the residual errors corresponding to the updated syndrome decrease. Bit Flipping decoders iterate this process and progressively build an estimate of the error vector. Many design choices are possible. For example, one may vary the number of bits flipped at each iteration or the strategy used to select them. The literature contains numerous BF decoder variants proposed for code-based KEMs [2, 4, 7]. These designs pursue different trade-offs among decoding capability, computational complexity, and the possibility of obtaining reliable DFR estimates. The importance of the last aspect is largely a consequence of the IND-CCA2 requirement discussed above. In ephemeral deployment scenarios, a rigorous DFR assessment is less critical. This opens the door to different trade-offs, where the focus shifts towards implementation efficiency and sufficiently low failure rates.

The recent proposal of BF-Max follows this line of reasoning [3, 19]. Starting from an all-zero estimated error vector, each iteration flips a single bit chosen so as to maximize the reduction of the syndrome weight. Although this strategy requires more iterations than conventional BF decoders, the aforementioned works show that it can achieve significantly lower DFRs than other BF variants reported in the literature. A straightforward implementation would be computationally prohibitive.

Table 2. Usability of HQC [15], BIKE [2], LeanBIKE [6], and our KEM instantiation across major UDP-based network protocols.

Scheme	**IKEv2** [14]	**DTLS 1.3** [18]	**QUIC** [13]
`HQC`	○	◑	◑
`BIKE`	○	◑	◑
`LeanBIKE`	○	●	◑
Ours	●	●	●

○ Not usable ◑ Usable, with frag. ● Usable, without frag.

However, taking sparsity into account makes it possible to achieve a theoretical complexity that is asymptotically comparable to that of other BF decoders. To the best of our knowledge, a practical implementation of BF-Max has never been presented. Therefore, it remains unclear whether BF-Max is mainly of theoretical interest or whether it can also become a practical and competitive decoding strategy for QC-MDPC code-based KEMs such as BIKE.

Contributions. This work investigates decoder design trade-offs for ephemeral QC-MDPC code-based Key Encapsulation Mechanisms and makes the following contributions.

- ▷ *Efficient implementation of BF-Max.* Practical implementations of BF-Max [3] have received limited attention in the QC-MDPC literature, despite the promising decoding-failure-rate results reported in recent works. In this work, we present an efficient AVX2 implementation of the BF-Max decoder, leveraging algorithmic-level optimizations and different data representation techniques to exploit the sparsity of the parity-check matrix along with data-parallel instructions offered by the AVX2 instruction set.
- ▷ *Hybrid decoding strategies with tunable DFR.* We propose a family of hybrid decoders for QC-MDPC codes that combine the error-correction capability of BF-Max with the efficiency of conventional parallel bit-flipping decoders. The framework we propose includes both switching strategies, where decoding starts with BF-Max and transitions to parallel bit flipping, and restart-based strategies employing multiple decoding attempts. These designs expose additional trade-offs between computational cost, execution time, and decoding-failure rate. We systematically analyse the resulting design space and identify decoder configurations offering tunable DFR characteristics for ephemeral KEM deployments.
- ▷ *Compact KEM instantiation.* We evaluate the proposed BF-Max implementation and hybrid decoding strategies by comparing their speed, object sizes, and decoding-failure rate against state-of-the-art QC-MDPC decoding algorithms. We then benchmark a complete ephemeral KEM instantiation built around the proposed decoder architecture. When compared to LeanBIKE, a code-based KEM specifically designed for ephemeral deployment scenarios, our design achieves a 7.8% reduction in bandwidth requirements while maintaining comparable execution times. We additionally evaluate the resulting parameter sets against established code-based KEMs designed to achieve conventional IND-CCA2 security targets. In this setting, the parameters we propose reduce the combined public-key and ciphertext size by up to 22% with respect to BIKE and by a factor of 2.79× with respect to the standardized KEM HQC. Table 2

summarizes the practical impact of these results on the deployment of code-based KEMs in bandwidth-constrained ephemeral key-establishment scenarios.

2 Notation and Background

In the following, we denote scalar variables in lowercase italic, x, or lowercase monospace, $\mathtt{x}$, when used in algorithmic descriptions. Vector quantities are denoted in boldface, e.g., $\mathbf{x}$. The i-th element of a vector $\mathbf{x}$ is denoted by $\mathbf{x}_i$, and its (Hamming) weight by $\mathrm{wt}(\mathbf{x})$.

The support of a vector $\mathrm{Supp}(\mathbf{x}) = \{i \colon \mathbf{x}_i \neq 0\}$ is the set of positions at which its entries are nonzero. A matrix is in uppercase bold, $\mathbf{X}$, with its i-th row denoted by $\mathbf{X}_{i,*}$ and its j-th column by $\mathbf{X}_{*,j}$. A set is denoted in uppercase italic, X, and $\mathbb{F}_2$ denotes the binary finite field.

A linear code of length n and dimension k is a k-dimensional linear subspace $\mathcal{C} \subseteq \mathbb{F}_2^n$. A linear code can be described by a full-rank matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, referred to as *parity-check matrix*, such that $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n : \mathbf{H}\mathbf{c}^\top = \mathbf{0}\}$. We denote by $\mathbf{H}_{i,*}$ the i-th row of $\mathbf{H}$, which defines the i-th parity-check equation, by $\mathbf{H}_{*,j}$ its j-th column, whose support identifies the parity-check equations involving the j-th code bit, and by $\mathbf{H}_{i,j}$ its entry at row i and column j. A vector $\mathbf{e} \in \mathbb{F}_2^n$ is said to be an *error vector* if $\mathbf{e} \notin \mathcal{C}$, and its *syndrome* is $\mathbf{s} = \mathbf{H}\mathbf{e}^\top$. By number of errors, we refer to $t = \mathrm{wt}(\mathbf{e})$. In this paper, decoding consists of an algorithm aiming to retrieve $\mathbf{e}$ from the knowledge of $\mathbf{H}$ and $\mathbf{s}$. Notice that this may not always be possible: in such a case, we experience a *decoding failure*.

QC-MDPC codes. An $r \times r$ matrix $\mathbf{A}$ is said to be circulant if it is of the form

$$\mathbf{A} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{r-1} \\ a_{r-1} & a_0 & \cdots & a_{r-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \cdots & a_0 \end{bmatrix}. \tag{1}$$

Circulant matrices over $\mathbb{F}_2$ form a commutative ring isomorphic to $\mathcal{R} = \mathbb{F}_2[x]/(x^r - 1)$ through the map $\mathbf{A} \leftrightarrow a(x) = \sum_{i=0}^{r-1} a_i x^i$, so that operations on circulant matrices reduce to polynomial arithmetic in $\mathcal{R}$. Similarly, a polynomial $a(x) \in \mathcal{R}$ can be associated with a vector rather than a matrix. For example, $\mathbf{a} \leftrightarrow a(x) = \sum_{i=0}^{r-1} a_i x^i$, where $\mathbf{a} = [a_0, a_1, \cdots, a_{r-1}]$ is the first row of $\mathbf{A}$.

In this paper, we focus on a special class of error correcting codes: QC-MDPC codes with parity-check matrices formed by a single row of circulant matrices.

Definition 1. *For $b \geq 2$, a QC-MDPC code of length $n = br$ and dimension $(b-1)r$ is a binary linear code with parity-check matrix $\mathbf{H} = [\mathbf{H}_0 \,|\, \cdots \,|\, \mathbf{H}_{b-1}] \in \mathbb{F}_2^{r\times n}$, where each $\mathbf{H}_i$ is an $r \times r$ circulant matrix of row and column weight v, with $v = O(\sqrt{r})$.*

In this paper, we will frequently focus on the special case $b = 2$, since it is the one employed in BIKE. Using the polynomial description, when $b = 2$ the parity-check matrix can be associated with a pair of polynomials $(h_0(x),\ h_1(x)) \in \mathcal{R}^2$, with $h_0(x)$ and $h_1(x)$ being the polynomials associated with the first row of $\mathbf{H}_0^\top$ and $\mathbf{H}_1^\top$, respectively. Notice that, according to the above definition, both $h_0(x)$ and $h_1(x)$ can be seen as very sparse polynomials, with very few coefficients set to 1. The syndrome of an error vector $\mathbf{e} = [\mathbf{e}_0 \,|\, \mathbf{e}_1] \in \mathbb{F}_2^n$ of weight t through $\mathbf{H}$ can then be computed as $\mathbf{s} = \mathbf{H}\mathbf{e}^\top$ and $\mathbf{s}^\top$ is described in polynomial terms as

$$s(x) = e_0(x)h_0(x) + e_1(x)h_1(x), \quad \mathrm{wt}(e_0(x)) + \mathrm{wt}(e_1(x)) = t.$$

Table 3. BIKE parameter sets [2].

NIST category	r	v	t	**Public key** (B)	**Ciphertext** (B)
1	12 323	71	134	1 541	1 573
3	24 659	103	199	3 083	3 115
5	40 973	137	264	5 122	5 154

When t is small as well, $s(x)$ is the sum of a few rotations of $h_0(x)$ and a few rotations of $h_1(x)$; this view will become rather handy later on. See Table 3 for examples of parameter sets we will consider.

2.1 The BIKE Key Encapsulation Mechanism

BIKE uses QC-MDPC codes with $b = 2$. The secret key consists of two polynomials $h_0(x), h_1(x) \in \mathcal{R}$ of odd weight v, describing the parity-check matrix $[\mathbf{H}_0 \,|\, \mathbf{H}_1]$ of the secret QC-MDPC code. The public key is $h(x) = h_0(x)^{-1}h_1(x) \in \mathcal{R}$, which describes the systematic parity-check matrix $[\mathbf{I}_r \,|\, \mathbf{H}_0^{-1}\mathbf{H}_1]$ of the same code and requires only r bits of storage. BIKE follows the Niederreiter framework, with a variant of the Fujisaki-Okamoto transform [11] used to achieve an IND-CCA secure Key Encapsulation Mechanism (KEM). To encapsulate, the sender chooses a random seed $\mathbf{m}$ and derives an error vector $\mathbf{e} = [\mathbf{e}^{(0)} \,|\, \mathbf{e}^{(1)}] \in \mathbb{F}_2^n$ of weight t from $\mathsf{Hash}(\mathbf{m}, h(x))$. The ciphertext, $\mathbf{c} = (\mathbf{c}^{(0)}, \mathbf{c}^{(1)})$, with $\mathbf{c}^{(0)} \in \mathbb{F}_2^r$, $\mathbf{c}^{(1)} \in \mathbb{F}_2^{256}$, and the shared key, $\boldsymbol{\kappa}$, are then computed as:

$$\begin{cases} \mathbf{c}^{(0)} = e_0(x) + e_1(x)h(x), \\ \mathbf{c}^{(1)} = \mathbf{m} \oplus \mathsf{Hash}(e_0(x), e_1(x)), \\ \boldsymbol{\kappa} = \mathsf{Hash}(\mathbf{m}, \mathbf{c}^{(0)}, \mathbf{c}^{(1)}), \end{cases} \tag{2}$$

where $[e_0(x), e_1(x)] \in \mathcal{R}^2$ is the pair of polynomials associated with the binary vector $\mathbf{e} = [\mathbf{e}^{(0)} \,|\, \mathbf{e}^{(1)}]$, $\oplus$ denotes bitwise addition and $\mathbf{c}^{(0)}$ is the syndrome of $\mathbf{e}$ with respect to the public matrix. To decapsulate, the recipient computes $s(x) = c_0(x)h_0(x) = e_0(x)h_0(x) + e_1(x)h_1(x)$, which corresponds to the transposed syndrome vector $\mathbf{s}^\top = \mathbf{e}\mathbf{H}^\top$ of $\mathbf{e}$ with respect to the secret matrix, and runs a QC-MDPC decoder on $\mathbf{H}$ to recover $\mathbf{e}$. It then computes $\mathbf{m} = \mathbf{c}_1 \oplus \mathsf{Hash}(\mathbf{e}^{(0)}, \mathbf{e}^{(1)})$ and, if $(\mathbf{e}^{(0)}, \mathbf{e}^{(1)})$ matches the error derived from $\mathsf{Hash}(\mathbf{m}, \mathbf{h})$, returns $\boldsymbol{\kappa} = \mathsf{Hash}(\mathbf{m}, \mathbf{c}^{(0)}, \mathbf{c}^{(1)})$; otherwise, including the case of a decoding failure, it returns $\mathsf{Hash}(\boldsymbol{\zeta}, \mathbf{c}^{(0)}, \mathbf{c}^{(1)})$, where $\boldsymbol{\zeta}$ is a random secret string stored in the private key. Table 3 shows BIKE parameters for the three NIST security categories.

Security. BIKE relies on the fact that while adversaries know only the systematic parity-check matrix of the QC-MDPC code (which is dense with high probability), the legitimate user instead knows a privileged representation, namely, the sparse parity-check matrix. Indeed, the sparse representation allows performing efficient decoding (e.g., using a Bit Flipping algorithm [8, 16]), while the dense representation requires to use generic decoders such as Information Set Decoding (ISD) [17], having execution time which is exponential in the value of t. Attackers may also try to recover the sparse parity-check matrix corresponding to the secret key, which boils down to the problem of finding low weight codewords. This can be accomplished, again, through ISD, which in this case takes time which is exponential in v. Therefore, the values of v and t must be chosen large enough, in order to avoid attacks. The quasi-cyclic structure only yields a polynomial speedup, in the order of $\sqrt{r}$ for message recovery (i.e., derivation of $\mathbf{e}$) and r for (secret) key recovery [21].

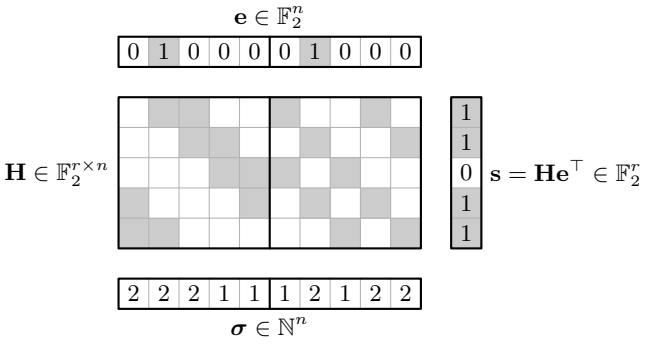


Fig. 1. Example of error vector, parity-check matrix, syndrome and resulting counters, for a QC-MDPC code with $r = 5$, $t = 2$, $b = 2$, $v = 2$, and $n = br = 10$.

2.2 Bit-Flipping Decoders

Decoders for QC-MDPC codes are derived from the bit-flipping decoder originally introduced by Gallager [8]. Given a syndrome $\mathbf{s}$, the decoder seeks to recover an error vector $\widehat{\mathbf{e}}$ such that $\mathbf{s} = \mathbf{H}\widehat{\mathbf{e}}^\top$. To this end, it starts from the all-zero estimate $\widehat{\mathbf{e}} = \mathbf{0}$ and iteratively refines it by identifying positions that are likely to contain errors. These decoders can be viewed as *statistical decoders*: rather than identifying erroneous positions deterministically, they estimate their likelihood of being in error from the information provided by the parity-check equations. The syndrome $\mathbf{s}$ indicates which parity checks are currently unsatisfied. Intuitively, a codeword position participating in many unsatisfied parity checks is more likely to be erroneous than a position involved mostly in satisfied ones. Algorithm 1 illustrates the general structure of the decoding procedure, while Figure 1 provides a visual representation of the objects involved in the decoder. At each iteration, a vector of counters $\boldsymbol{\sigma}$ is computed, where each value $\boldsymbol{\sigma}_j$ $(0 \leq j \leq n-1)$ equals the number of unsatisfied parity checks involving the j-th codeword position or, equivalently, the amount of ones shared between the j-th column of the parity-check matrix $\mathbf{H}$ and the current syndrome $\mathbf{s}$. Since $\mathbf{H}$ is sparse, positions corresponding to columns with a high overlap with $\mathbf{s}$, i.e., positions with larger counters, are considered more likely to be erroneous. The set J of positions selected for correction is then derived from the counter vector according to a chosen flipping rule. Different choices for this rule lead to different decoding algorithms, which will be discussed later. For every position $j \in J$, the corresponding component of the error estimate is flipped, i.e., $\widehat{\mathbf{e}}_j \leftarrow \widehat{\mathbf{e}}_j + 1$, and the syndrome is updated accordingly as $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,j}$. The decoding process terminates when the syndrome becomes zero or when the maximum number of iterations is reached. Based on how flipping decisions are performed, and when the syndrome and the counter vector are updated during the decoding process, bit-flipping decoders can be classified into two main categories:

- *Sequential* (sometimes also called step-by-step or in-place): the decoder performs one flip at a time, updating the syndrome and all counters after each flip (i.e., $|J| = 1$);
- *Parallel* (sometimes also called out-of-place): multiple bits are flipped before the syndrome and counters are updated (i.e., $|J| > 1$).

The decoders employed by BIKE within the NIST PQC standardization process, namely, BGF and BIKE-Flip [2, 7], follow the parallel approach. Their structure is particularly well suited for effi-

Algorithm 1: Bit-Flipping Decoder

Data: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, maximum number of iterations $\texttt{itMax} \in \mathbb{N}$
Input: syndrome $\mathbf{s} \in \mathbb{F}_2^r$
Output: decoding failure $\perp$, or vector $\widehat{\mathbf{e}} \in \mathbb{F}_2^n$ such that $\mathbf{s} = \mathbf{H}\widehat{\mathbf{e}}^\top$

$\widehat{\mathbf{e}} \leftarrow \mathbf{0}$
$\texttt{iter} \leftarrow 0$
while $(\mathbf{s} \neq \mathbf{0}) \wedge (\texttt{iter} < \texttt{itMax})$ **do**
  for $j = 0, \cdots, n-1$ **do**
    compute $\boldsymbol{\sigma}_j$
  determine $J \subseteq \{0, \cdots, n-1\}$ so that, for each $j \in J$, $\boldsymbol{\sigma}_j$ is large enough
  for $j \in J$ **do**
    $\widehat{\mathbf{e}}_j \leftarrow \widehat{\mathbf{e}}_j + 1$ // XOR
    $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,j}$ // elementwise XOR
  $\texttt{iter} \leftarrow \texttt{iter} + 1$
if $\mathbf{s} \neq \mathbf{0}$ return $\perp$, **else** return $\widehat{\mathbf{e}}$

cient implementation, as the flipping decisions can be performed in parallel and efficiently mapped with SIMD instructions, resulting in low decoding latency.

BF-Max. Recently, a novel sequential decoder called BF-Max has been introduced [3, 19], whose pseudocode is shown in Algorithm 2. As a peculiarity, in this decoder the set J is determined as the singleton containing the index pointing at one of the bits with maximum counter, i.e., $J = \{\arg\max\{\boldsymbol{\sigma}\}\}$. If there are multiple bits with the same (maximum) counter value, the decoder chooses one of these bits uniformly at random. Despite its promising decoding properties, BF-Max has not, to the best of our knowledge, been extensively investigated as a practical decoder integrated into a complete KEM.

3 Decoder Implementation

In this section, we describe our optimized implementation of the BF-Max decoder. In Section 3.1, we focus on how to minimize the computational overhead given by the update of $\boldsymbol{\sigma}$ at each iteration of BF-Max, introducing an algorithmic improvement based on the $\mathbb{Z}^{b \cdot r \times b \cdot r}$ correlation matrix $\boldsymbol{\Gamma} = \mathbf{H}^\top \cdot \mathbf{H}$. In Section 3.2, we list the data structures we employ to represent binary vectors. Finally, in Section 3.3 we describe our optimized implementation of all the subroutines of BF-Max with the AVX2 instruction set.

3.1 Correlation Matrix

The main challenge of efficiently implementing BF-Max is to find the right way to exploit the sparsity of the $\mathbf{H}$ matrix to perform the most computationally expensive step of BF-Max, namely, the update of $\boldsymbol{\sigma}$ after each flip, in an efficient way. This is not straightforward, as every update is made of irregular memory accesses, which cannot be vectorized. Our key observation is that the counter update can be decomposed into a *fixed* part, which depends on the structure of the parity-check matrix, and a *variable* part, which depends on the flipped position and on the state of the

Algorithm 2: BF-Max

Data: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, maximum number of iterations $\texttt{itMax} \in \mathbb{N}$
Input: syndrome $\mathbf{s} \in \mathbb{F}_2^r$
Output: decoding failure $\perp$, or vector $\widehat{\mathbf{e}} \in \mathbb{F}_2^n$ such that $\mathbf{s} = \mathbf{H}\widehat{\mathbf{e}}^\top$

$\widehat{\mathbf{e}} \leftarrow 0$
$\texttt{iter} \leftarrow 0$
for $i = 0, \cdots, n-1$ **do**
  compute $\boldsymbol{\sigma}_i$
while $(\mathbf{s} \neq \mathbf{0}) \wedge (\texttt{iter} < \texttt{itMax})$ **do**
  $\texttt{pos} \leftarrow \text{argmax}\{\boldsymbol{\sigma}\}$
  $\widehat{\mathbf{e}}_{\texttt{pos}} \leftarrow \widehat{\mathbf{e}}_{\texttt{pos}} + 1$
  for $j \in \text{Supp}(\mathbf{H}_{*,\texttt{pos}})$ **do**
    $\mathbf{s}_j \leftarrow \mathbf{s}_j + 1$
    if $\mathbf{s}_j = 0$, set $d = -1$; **else** set $d = 1$
    for $k \in \text{Supp}(\mathbf{H}_{j,*})$ **do**
      $\boldsymbol{\sigma}_k \leftarrow \boldsymbol{\sigma}_k + d$
  $\texttt{iter} \leftarrow \texttt{iter} + 1$
if $\mathbf{s} \neq \mathbf{0}$ return $\perp$, **else** return $\widehat{\mathbf{e}}$

syndrome. To explain why this is the case, we define the $\mathbb{Z}^{b \cdot r \times b \cdot r}$ *correlation matrix* $\boldsymbol{\Gamma} := \mathbf{H}^\top \cdot \mathbf{H}$, where the product is computed over the integers by first lifting the entries of parity-check matrix $\mathbf{H}$ from $\mathbb{F}_2$ to $\mathbb{Z}$. We have that the entry (i, j) of $\boldsymbol{\Gamma}$ corresponds to the number of parity checks shared by the i-th and j-th error coordinate, i.e., $\boldsymbol{\Gamma}_{i,j} = |\,\text{Supp}(\mathbf{H}_{*,i}) \cap \text{Supp}(\mathbf{H}_{*,j})|$. A visual representation of the correlation matrix is provided in Figure 2. When a flip is performed in position i, the counter in position j, $\boldsymbol{\sigma}_j$, is updated a total of $\boldsymbol{\Gamma}_{i,j}$ times, where each update is either 1 or -1 depending on the change in the corresponding syndrome bit. Indeed, after the update $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,i}$, for each of the $\boldsymbol{\Gamma}_{i,j}$ shared parity-checks that become satisfied $(1 \to 0)$ the counter is decreased by 1, and for each parity-check that becomes unsatisfied $(0 \to 1)$ the counter is increased by 1.

This set of updates can be performed in an alternative way, by first decreasing $\boldsymbol{\sigma}_j$ by the maximum possible amount ($\boldsymbol{\Gamma}_{i,j}$), and then increasing it by $2\times$ the number of shared parity-checks that become unsatisfied. If the number of parity checks that become unsatisfied is small w.r.t. v (the total number of bits changed in the update $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,i}$), then the cost of updating $\boldsymbol{\sigma}$ can be reduced by first adding $-\boldsymbol{\Gamma}_{i,j}$ to all the counters $\boldsymbol{\sigma}_j$, $0 \leq j \leq n-1$, and then increasing by 2 only the counter positions that are included in the few parity checks that become unsatisfied. In the following, we denote the portion of -1 updates as *majority* group, and the portion of $+1$ updates as *minority* group. Our technique allows the updates in the *majority* group to be performed in a single step, with the use of the $\boldsymbol{\Gamma}$ matrix.

Out of the v rows of $\mathbf{H}$ that are involved in updating $\boldsymbol{\sigma}$ during an iteration, at least $\left\lfloor \frac{v}{2} \right\rfloor + 1$ are expected to end up in the *majority* group. As a consequence, using $\boldsymbol{\Gamma}$ allows us to reduce the number of scattered updates by a factor $\geq \frac{2v}{v+1}$ (replacing *at least* $\left\lfloor \frac{v}{2} \right\rfloor + 1$ out of v updates with a single one). In theory it is also possible for a BF-Max iteration to cause more parity checks to become unsatisfied $(0 \to 1)$ than satisfied $(1 \to 0)$. In this case the *majority* group would represent the set of precomputed *sums* to be applied to $\boldsymbol{\sigma}$, instead of subtractions. However, this would likely

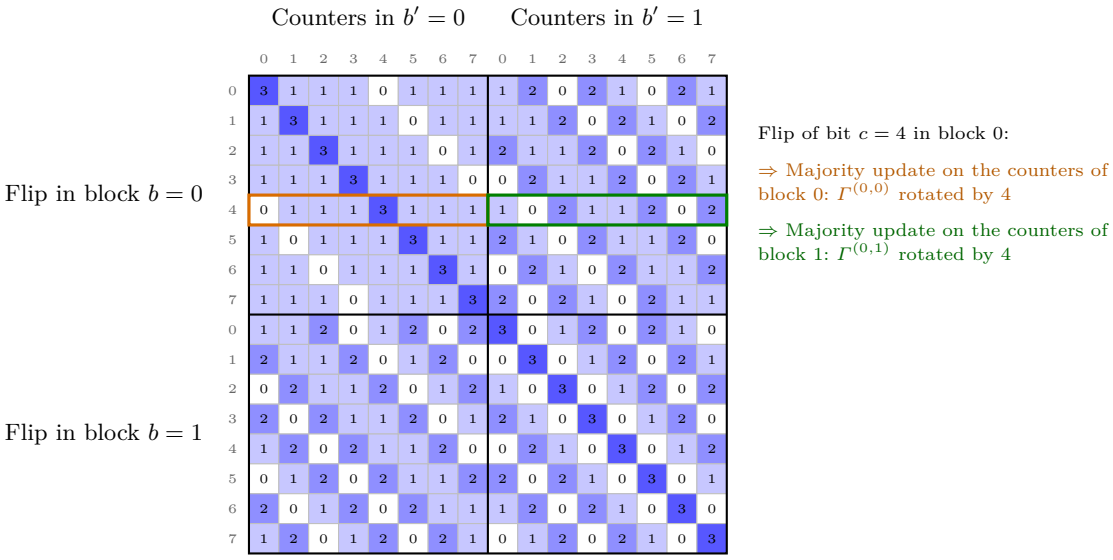


Fig. 2. Correlation matrix $\boldsymbol{\Gamma}$ for a toy QC-MDPC code ($b = 2$, $r = 8$, $v = 3$), with $\mathrm{Supp}(\mathbf{H}_0) = \{0, 1, 3\}$, $\mathrm{Supp}(\mathbf{H}_1) = \{0, 2, 5\}$.

be an indication that the iteration has selected the wrong bit of $\hat{\mathbf{e}}$ to flip, since $\mathrm{wt}(\mathbf{s})$ has increased instead of going down.

We point out that the correlation matrix $\boldsymbol{\Gamma}$ is symmetric, and composed of $b \times b$ circulant blocks of size $r \times r$. We can exploit this fact to store in memory only the first row $\boldsymbol{\Gamma}^{(b,b')}$ of each circulant block: the rest of the matrix can be obtained by means of block-wise rotations of such row. Moreover, since $\boldsymbol{\Gamma}$ depends only on $\mathbf{H}$, it can be computed once during key generation and stored with the secret key. However, in our implementation employed for comparisons with other decoding techniques, we compute it at the beginning of every call to the decoder, so that its cost is included in the decoding times we report and therefore the performance comparison with other decoders not relying on precomputed data is fair.

3.2 Binary Vector Representations

The error vector $\mathbf{e}$, the rows and columns of $\mathbf{H}$, and the syndrome $\mathbf{s}$ are all low-weight binary vectors. To represent them, we distinguish between three possible data structures, each presenting its own tradeoffs depending on the type of SIMD access pattern required to read and update it. Given a vector $\mathbf{v}$ of Hamming weight w and bit-length l, we employ one or more of the following representations:

- **Sparse**: an array holding the indexes of all the active bits in $\mathbf{v}$. The sequence of indexes does not need to be ordered. The data type used for the elements must allow counting from 0 up to the last index $l - 1$; in our case the choice always falls on `uint32_t`. The memory footprint of this representation is $32 \cdot w$ bits.

- **Dense**: a plain bit sequence, i.e., an array of 64-bit words which, when read in order, form the bits of $\mathbf{v}$. In memory this takes up $64 \cdot \lceil \frac{l}{64} \rceil$ bits (the nearest multiple of 64 greater than or equal to l).
- **Bit-expanded**: each bit of $\mathbf{v}$ is stored as a separate 8-bit integer (`uint8_t`), where the 7 most significant bits are always 0. This is the most memory-expensive representation, occupying $8 \cdot l$ bits.

In the following section, we show how choosing different data structures can make the vectors faster to access and update.

3.3 AVX2 Implementation

The BF-Max algorithm shown in Algorithm 2 first initializes $\hat{\mathbf{e}}$ to the all-zero vector, then computes the vector of unsatisfied parity checks $\boldsymbol{\sigma}$, and finally performs a certain number of decoding iterations. During each iteration, the position with the highest value in the $\boldsymbol{\sigma}$ vector is selected, and the corresponding bit of the error vector is flipped, which in turn causes both the syndrome and $\boldsymbol{\sigma}$ itself to be updated. The procedure stops in two cases: as soon as the syndrome becomes an all-zero vector (decoding success), or when a predefined maximum number of iterations `itMax` is reached (decoding failure). Since the Hamming weight of $\mathbf{e}$ is a fixed parameter of the cryptosystem ($\mathrm{wt}(\mathbf{e}) = t$), and only one bit of $\hat{\mathbf{e}}$ is flipped at each decoding iteration, the algorithm must perform at least t iterations to succeed ($\texttt{itMax} \geq t$). Whenever the decoding procedure terminates successfully, the output is the desired error vector $\mathbf{e}$.

$\boldsymbol{\sigma}$ initialization. The vector of unsatisfied parity checks first needs to be initialized (line 4 of Algorithm 2) at the beginning of the decoding process. To store $\boldsymbol{\sigma}$ we can use an n-element array of 8-bit integers, since the highest value a counter can reach is v, which is always lower than 2^8. Each counter $\boldsymbol{\sigma}_i$ can be thought of as $\mathrm{wt}(\mathbf{H}_{*,i} \wedge \mathbf{s})$, the Hamming weight of the bitwise `and` operation between the i-th column of $\mathbf{H}$ and the syndrome. To count this weight efficiently, we materialize each column of $\mathbf{H}$ in sparse form (so we can enumerate the positions of its non-zero bits sequentially), and also store $\mathbf{s}$ bit-expanded. This allows us, for every non-zero bit of the column, to immediately check whether the corresponding syndrome bit is set, in which case we increment the counter $\boldsymbol{\sigma}_i$ without the need for bit-extraction masks. We take advantage of the circulant nature of the blocks of $\mathbf{H}$ by only storing the first column of each block. The other columns in a block are cyclic shifts of the first, and can be obtained by incrementing the values of the sparse representation mod r.

$\boldsymbol{\sigma}$ update. The heaviest operation in terms of computational complexity is updating the entries of the $\boldsymbol{\sigma}$ vector (lines 8 to 12 of Algorithm 2). Each decoding iteration selects a single bit of $\mathbf{e}$ to flip, say at index i, which immediately causes v bits of $\mathbf{s}$ to flip (positioned as the active bits in column $\mathbf{H}_{*,i}$). Then, each modified bit of $\mathbf{s}$ influences $b \cdot v$ entries of $\boldsymbol{\sigma}$, because flipping bit j in $\mathbf{s}$ alters all the counters corresponding to the active bits in row $\mathbf{H}_{j,*}$, which are $\mathrm{wt}(\mathbf{H}_{j,*}) = b \cdot v$. In total, a decoding iteration performs $v^2 \cdot b$ updates.

As introduced in Section 3.1, we can accelerate these updates with the help of the correlation matrix $\boldsymbol{\Gamma}$, by bulk-decrementing $\boldsymbol{\sigma}$ at fixed positions, which depend on $\mathbf{H}$ alone, and then incrementing only the counter positions corresponding to the few parity checks that become unsatisfied (the *minority* group). As pointed out, $\boldsymbol{\Gamma}$ is composed of b^2 circulant blocks, so we can precompute only the first row of each block once, and obtain the other rows via rotation. Each element $\boldsymbol{\Gamma}_{i,j}$ is represented as an 8-bit integer to be summed to counter $\boldsymbol{\sigma}_j$, so the memory footprint of the correlation matrix is $b^2 \cdot r$ bytes. During a BF-Max iteration we perform the updates in the *majority*

group by repeatedly selecting rows of $\boldsymbol{\Gamma}$ and subtracting them element-wise from $\boldsymbol{\sigma}$. These are all subtractions between two n-long vectors of 8-bit elements, which can easily be vectorized with the `vpsubb` instruction.

To take care of the updates in the *minority* group instead, we necessarily need to access the rows of $\mathbf{H}$ to check, at every iteration, which parity check equations have become unsatisfied ($0 \to 1$ in $\mathbf{s}$) after flipping a bit of $\mathbf{e}$. To materialize the necessary rows efficiently we can again take advantage of their cyclic nature. We only store the first row of each circulant block, in sparse form, then rotate it as needed using SIMD operations. Operationally, from the sparse representation of the first row, we can obtain the i-th row of the same block by adding i to every element of its support and then reducing mod r. Using three AVX2 instructions, this modular addition can be performed in parallel on 8 elements at a time (each 256-bit register can fit 8 32-bit elements of the sparse row). The sequence of instructions is: `vpaddd`, `vpsubd`, `vpminud`. The first instruction sums i elementwise to the sparse representation of the first row, producing results in the range $[0, 2r)$, of which only the ones $\geq r$ need to be reduced. The second subtracts r from the previous result, obtaining either the correct reduced result or a very large integer by underflow. Finally, `vpminud` selects the correct result, discarding the larger value of the two.

A further optimization stems from the observation that we can buffer the updates to $\boldsymbol{\sigma}$, instead of performing them immediately. After finding a counter to update in the *minority* group, we add it to a queue and then update $\boldsymbol{\sigma}$ in bulk before moving on to the next decoding iteration. This enables better cache locality, accelerating the most computationally heavy step of the decoder. We note that the updates in the *minority* must actually be performed before the ones in the *majority* group, to avoid underflows.

The updates to $\mathbf{e}$ and $\mathbf{s}$ (lines 7 and 9 of Algorithm 2, respectively) are straightforward provided that we choose the appropriate binary vector representations. For $\mathbf{e}$, only one position is flipped per iteration, so we can use a dense representation, which favors this type of rare and scattered update: the 64-bit word at the correct index is accessed and one of its bits is flipped via XOR. The syndrome instead is updated v times per iteration. Storing it in bit-expanded form (as done during the $\boldsymbol{\sigma}$ initialization) allows us to access and update its bits quickly, with a small penalty in memory consumption.

Throughout the algorithm we access both the rows and columns of $\mathbf{H}$. In theory it is possible to store only the first row of each block of $\mathbf{H}$, and obtain all the necessary rows and columns from those (or, vice versa, to store only the first columns). However, this extra computation results in a considerable time penalty, with only a small saving in memory, so we opt to store both the first row and the first column of each block. Similarly, precomputing the whole sparse matrix saves the clock cycles needed for the vector rotations, but ultimately results in a slower implementation due to cache misses (per block, we would need to store $r \cdot v$ positions instead of just v).

Computation of `argmax`. The `argmax` function at line 6, which finds the highest out of n values in $\boldsymbol{\sigma}$, is computed at every iteration. We implement such procedure by performing two full scans of $\boldsymbol{\sigma}$: one to find the maximum value and a second one to find its position. The first scan is performed concurrently with the majority updates over $\boldsymbol{\sigma}$, and thus requires no additional load operation. A single AVX2 register holds 32 8-bit counters, so $\boldsymbol{\sigma}$ is loaded 32 values at a time. A running vector of partial maxima, initialized with all zeros, is updated via the `vpmaxub` instruction, and its 32 lanes are reduced at the end of the first scan to a single byte, containing the largest value of $\boldsymbol{\sigma}$, with five shuffle-and-`vpmaxub` steps. Figure 3 shows a toy example of such process, acting on a vector $\boldsymbol{\sigma}$ of length 24. During the second scan, to find the correct position to be flipped, 32 counters are compared at once against the found maximum value using the `vpcmpeqb` instruction. The result is

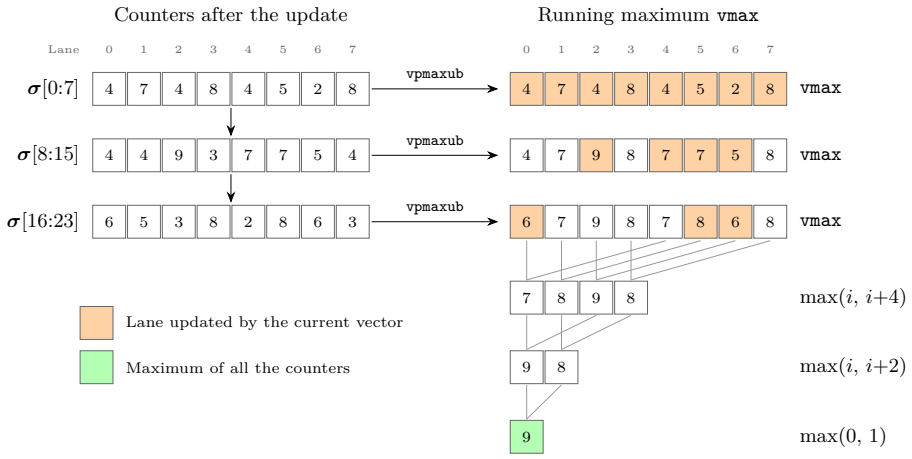


Fig. 3. Visual representation of the vector scan performed to compute $\max\{\boldsymbol{\sigma}\}$. The figure shows 8 counters in each register for simplicity; our AVX2 implementation stores 32 counters per register.

collapsed into a 32-bit mask with `vpmovmskb` and, when the maximum is found, the index within the AVX2 register of the first set bit is extracted with a count-trailing-zeros instruction. While stopping at the first match does not match the uniformly random tie-breaking rule of BF-Max, it does not alter error correction capabilities.

4 Decoder Algorithmic Design

In this section, we describe a set of techniques aimed at improving the performance, in terms of decryption speed and object size, of QC-MDPC based KEMs. First, we describe which metrics are relevant and what we want to improve in our analysis (Section 4.1), and then we propose different algorithmic improvements on existing iterative decoders (Section 4.2).

4.1 Design Tradeoff

When working with the decoding (and thus the decryption) primitive of a QC-MDPC based scheme, there are three principal metrics that define the performance of a specific parameter set and decoder choice: the Decoding Failure Rate (DFR), the size of the circulant blocks r, and the decoder speed.

Decoding Failure Rate (DFR). Iterative decoders such as BF and BF-Max exhibit a non-zero probability of decoding failures. When constructing a KEM targeting IND-CCA2, such probability must be low enough to prevent an active attacker from observing even a single failure in the whole keypair lifetime (e.g., $\leq 2^{-\lambda}$, where λ is the target security level) [9, 10, 23]. When considering security requirements for ephemeral KEMs, as in the case for IND-CPA or IND-1CCA, decoding failures do not affect the security of the scheme, but rather its reliability. In LeanBIKE [6], the authors consider a simplified decoding procedure and explore DFR levels in $\{10^{-5}, 10^{-6}, 10^{-7}\}$, much higher than the ones necessary for IND-CCA2 security.

Circulant block size r. The size of the circulant blocks composing the public and private keys has an impact on both the size of the transmitted object during the key-exchange, and the reliability

level of the KEM. Indeed, the number of bytes required to complete an ephemeral key-exchange is $\lceil \frac{r}{8} \rceil$ for the public key and $\lceil \frac{r}{8} \rceil$ for the ciphertext ($\lceil \frac{r}{8} \rceil + 32$ in the case of BIKE). On the other hand, reducing r causes the secret QC-MDPC matrix employed during decryption to become more dense (as the Hamming weight of its column, v, remains constant), worsening the decoding performances and increasing the DFR of the scheme. Finally, tuning r has a minimal influence on static Message Recovery Attacks (MRA) and Key Recovery Attacks (KRA), since the complexity of solving the underlying hard problems (Quasi-Cyclic Syndrome Decoding and Quasi-Cyclic Codeword Finding, respectively) depends polynomially on r [17, 21, 22].

Decoder Speed. The family of heuristic iterative decoders includes a heterogeneous set of algorithms, each providing different performance profiles. Algorithms such as the *parallel* BF decoder are designed to be efficient in terms of speed, especially when data parallelism can be exploited on the target machine. More tailored decoding strategies such as BF-Max allow a significant improvement in decoding effectiveness (i.e., in DFR), but require a larger amount of computation. Even for a fixed algorithm, such as the BF decoder, the choice of internal parameters such as the maximum number of iterations affects both the decoding speed and effectiveness.

When selecting the parameters and decoder for a QC-MDPC based scheme, one has to take these three factors into account, prioritizing speed, object size, or reliability according to the target application. In the next section, we propose improvements on the decoding strategy that allow for a smaller value of the circulant block r, and thus smaller public key and ciphertext size, while minimizing the computational overhead required to maintain low enough DFR levels.

4.2 Hybrid BF Decoders

The current implementation of QC-MDPC based cryptosystems employs, at its core, variations of the parallel bit-flipping decoder. In such decoding algorithm, the counters vector σ is computed once per iteration, and *all* the flipping decisions are performed simultaneously, before updating the syndrome. This strategy exploits parallelism to achieve good performance in terms of execution time, but is not optimal from the perspective of error correction capability. Whenever the number of errors t is large, or the size of the circulant block r is small, the statistical noise in the counters causes the parallel decoder to perform a large number of incorrect flips.

On the other hand, BF-Max is designed as a more conservative decoder. Indeed, flipping only the bit with the highest counter allows to reduce the number of discrepancies between the original error vector and its estimate, heuristically minimizing the probability of making incorrect flips. In terms of execution time, BF-Max exploits the sparsity of the parity-check matrix to speed up the updates on the vector $\boldsymbol{\sigma}$ after each flip. Nevertheless, BF-Max is somewhat less efficient than a parallel decoder when considering the fact that *at least* t iterations must be performed to achieve a decoding success (i.e., $\mathtt{itMax}_{\mathsf{BF\text{-}Max}} \geq t$). We can exploit the benefits of both strategies by combining the two decoders as shown in Algorithm 3. At the beginning of the decoding process, we employ the BF-Max procedure for a limited number of iterations $0 \leq \mathtt{itMax}_{\mathsf{BF\text{-}Max}} \leq t$, as it allows to correct some of the errors when their amount is large. After reducing the number of discrepancies, we can safely adopt the parallel decoder to correct the remaining errors efficiently. Combining the BF-Max and parallel decoders in this order improves error correction capabilities with respect to the sole parallel decoder, while mitigating the computational overhead introduced by BF-Max by employing it only in the error regime where it is necessary.

If a sufficient number of parallel iterations are performed in the second part of the decoding procedure, then the DFR of the resulting hybrid decoder is mainly driven by the error correcting

Algorithm 3: BFH($\texttt{itMax}_{\text{BF-Max}}, \texttt{itMax}_{par}$)

Data: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, iterations of BF-Max $\texttt{itMax}_{\text{BF-Max}} \in \mathbb{N}$, maximum number of parallel decoder iterations $\texttt{itMax}_{par} \in \mathbb{N}$, parallel decoder thresholds $\texttt{th}_i$ $(0 \leq i < \texttt{itMax}_{par})$
Input: syndrome $\mathbf{s} \in \mathbb{F}_2^r$
Output: decoding failure $\perp$, or vector $\widehat{\mathbf{e}} \in \mathbb{F}_2^n$ such that $\mathbf{s} = \mathbf{H}\widehat{\mathbf{e}}^\top$

Set $\widehat{\mathbf{e}} = \mathbf{0}$

// BF-Max iterations
$\texttt{iter} \leftarrow 0$
for $i = 0, \cdots, n-1$ **do**: compute $\boldsymbol{\sigma}_i$
while $\left(\mathbf{s} \neq \mathbf{0}\right) \wedge (\texttt{iter} < \texttt{itMax}_{\text{BF-Max}})$ **do**
  $\texttt{pos} \leftarrow \arg\max\{\boldsymbol{\sigma}\}$
  $\widehat{\mathbf{e}}_{\texttt{pos}} \leftarrow \widehat{\mathbf{e}}_{\texttt{pos}} + 1$; $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,\texttt{pos}}$
  update $\boldsymbol{\sigma}$
  $\texttt{iter} \leftarrow \texttt{iter} + 1$

// Parallel decoder iterations
$\texttt{iter} \leftarrow 0$
while $\left(\mathbf{s} \neq \mathbf{0}\right) \wedge (\texttt{iter} < \texttt{itMax}_{par})$ **do**
  for $i = 0, \cdots, n-1$ **do**: compute $\boldsymbol{\sigma}_i$
  for $i = 0, \ldots, n-1$ **do**
    if $\sigma_i \geq \texttt{th}_{\texttt{iter}}$ **then**
      $\widehat{\mathbf{e}}_i \leftarrow \widehat{\mathbf{e}}_i + 1$; $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,i}$
  $\texttt{iter} \leftarrow \texttt{iter} + 1$

if $\mathbf{s} \neq \mathbf{0}$ return $\perp$, **else** return $\widehat{\mathbf{e}}$

performance of the BF-Max in its first iterations. Indeed, if the first flips of BF-Max are correct, then the decoding instance is likely to succeed. On the other hand, if the first flips are incorrect (i.e., they introduce discrepancies instead of removing them), they inevitably lead to a cascade of incorrect flipping decisions and thus to a decoding failure. We can detect if such event is happening by looking at the value of $\boldsymbol{\sigma}_{pos}$ at each iteration of BF-Max.

- If the first flips are correct, then the number of discrepancies decreases. This causes the statistical noise in the counters $\boldsymbol{\sigma}$ to vanish, meaning that the values in the error-free positions get smaller and the values in the erroneous positions get larger;
- If the first flips are incorrect, then the number of discrepancies increases. This causes the noise in the counters $\boldsymbol{\sigma}$ to worsen, meaning that the values in the error-free and erroneous positions become harder to distinguish, by converging towards $\frac{v}{2}$.

This implies that the sequence of $\boldsymbol{\sigma}_{pos}$ at each iteration is increasing if the sequence of flips is correct, and decreasing or stationary if the sequence of flips is incorrect. By setting a lower bound $\texttt{th}_{restart}$ on the extracted maximum counter, we can successfully detect the second case, and restart the decoding process with a *different* initial flip.

Algorithm 4 shows our implementation of such procedure, with the additions with respect to Algorithm 3 highlighted in blue. During the first iteration of BF-Max, we keep track of the selected bit, in order to exclude it in case a restart is triggered. This strategy allows us to filter out most of

Algorithm 4: BFH-R($\texttt{itMax}_{\text{BF-Max}}, \texttt{itMax}_{par}$)

Data: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, iterations of BF-Max $\texttt{itMax}_{\text{BF-Max}} \in \mathbb{N}$, maximum number of parallel decoder iterations $\texttt{itMax}_{par} \in \mathbb{N}$, parallel decoder thresholds $\texttt{th}_i$ $(0 \leq i < \texttt{itMax}_{par})$, maximum restarts `resMax`, restart counter threshold $\texttt{th}_{restart}$
Input: syndrome $\mathbf{s} \in \mathbb{F}_2^r$
Output: decoding failure $\perp$, or vector $\widehat{\mathbf{e}} \in \mathbb{F}_2^n$ such that $\mathbf{s} = \mathbf{H}\widehat{\mathbf{e}}^\top$

Set $\widehat{\mathbf{e}} = \mathbf{0}$
iter $\leftarrow 0$; restarts $\leftarrow 0$; excluded $\leftarrow \emptyset$
while $(\texttt{iter} < \texttt{itMax}_{BF\text{-}Max}) \wedge (\texttt{restarts} < \texttt{resMax})$ **do**
  // First BF-Max iteration
  iter $\leftarrow 0$
  for $i = 0, \cdots, n-1$ **do**: compute $\boldsymbol{\sigma}_i$
  pos $\leftarrow \arg\max_i \{\sigma_i \mid i \notin \texttt{excluded}\}$
  $\widehat{\mathbf{e}}_{\text{pos}} \leftarrow \widehat{\mathbf{e}}_{\text{pos}} + 1$; $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,\text{pos}}$
  excluded $\leftarrow$ excluded $\cup$ {pos}
  update $\boldsymbol{\sigma}$
  iter $\leftarrow$ iter $+ 1$
  while iter $< \texttt{itMax}_{BF\text{-}Max}$ **do**
    // From second to last BF-Max iteration
    pos $\leftarrow \arg\max\{\boldsymbol{\sigma}\}$
    if $\boldsymbol{\sigma}_{\text{pos}} \leq \texttt{th}_{restart}$ **then**
      Reset $\widehat{\mathbf{e}}$ and $\mathbf{s}$
      restarts $\leftarrow$ restarts $+ 1$
      break
    $\widehat{\mathbf{e}}_{\text{pos}} \leftarrow \widehat{\mathbf{e}}_{\text{pos}} + 1$; $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,\text{pos}}$
    update $\boldsymbol{\sigma}$
    iter $\leftarrow$ iter $+ 1$
iter $\leftarrow 0$
while $\left(\mathbf{s} \neq \mathbf{0}\right) \wedge (\texttt{iter} < \texttt{itMax}_{par})$ **do**
  for $i = 0, \cdots, n-1$ **do**: compute $\boldsymbol{\sigma}_i$
  for $i = 0, \ldots, n-1$ **do**
    if $\boldsymbol{\sigma}_i \geq \texttt{th}_{\texttt{iter}}$ **then**
      $\widehat{\mathbf{e}}_i \leftarrow \widehat{\mathbf{e}}_i + 1$; $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{H}_{*,i}$
  iter $\leftarrow$ iter $+ 1$
if $\mathbf{s} \neq \mathbf{0}$ return $\perp$, **else** return $\widehat{\mathbf{e}}$

the decoding failures due to a cascade of incorrect flipping decisions, improving the error correction capabilities of the decoder. From a computational performance perspective, the overhead introduced by the restarts is negligible in regimes where the DFR of the sole hybrid decoder is sufficiently small (e.g., less than 10^{-2}), since the amount of instances where a restart is required is low enough to not alter the average execution time in a significant way.

5 Experimental Results

In this section, we present the results of our performance evaluation campaign. We test the performance, in terms of speed, required block size and DFR, of BF-Max and the family of hybrid decoders presented in Section 4, implemented according to the techniques described in Section 3. We instantiate such decoders with different values for $\texttt{itMax}_{\text{BF-Max}}$ and $\texttt{itMax}_{par}$, to explore the tradeoff between runtime and error correction capabilities. The benchmarks are run on a Debian 12 system with an AMD EPYC 9654 CPU clocked at 2.40 GHz, frequency scaling enabled, and execution pinned to a single core. The clock cycles are counted via the `rdtscp` instruction and averaged over 10^6 runs. We used `gcc` version 12.2.0 and compiled the test binaries with the `-O3 -mavx2` flags. In our KEM analysis, we consider parameter sets targeting 128 bits of security (corresponding to NIST security level 1), since higher security levels are unlikely to fit in bandwidth-constrained protocols.

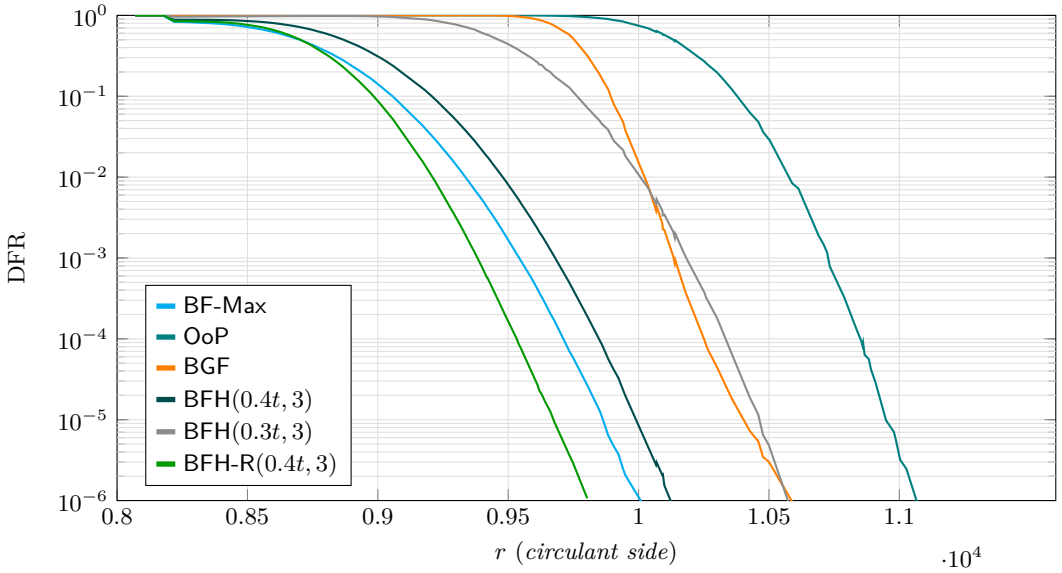


Fig. 4. Decoding failure rate of QC-MDPC decoders with $b = 2$, $v = 71$, and $t = 134$. The DFR is shown as a function of r.

Figure 4 shows the decoding failure rate of several decoders, acting on errors with Hamming weight $t = 134$, and QC-MDPC codes having $v = 71$ (as in BIKE and LeanBIKE), with different values of r (i.e., the size of the circulant blocks). These figures were obtained by running 10^8 simulations, for each decoder, on randomized inputs. We compare the BF-Max decoder with $\texttt{itMax}_{\text{BF-Max}} = 1.5t$, our proposed hybrid decoder instantiated with various iteration caps (BFH$(0.4t, 3)$ and BFH$(0.3t, 3)$), and our hybrid decoder *with restarts* (BFH-R$(0.4t, 3)$), against the standard parallel bit flipping decoder (OoP) with $\texttt{itMax}_{par} = 3$, and the three-iteration Black-and-Gray Flip decoder (BGF) currently employed in the official LeanBIKE implementation. The additional parameters required by the BFH, BFH-R, and OoP decoders, such as the flipping thresh-

olds employed in the parallel iterations, were found by means of a simple exhaustive search before running the DFR and timing tests. To ensure a fair comparison between different decoding strategies, we relax constant-time constraints also for the OoP and BGF decoders.

As can be seen, both the BF-Max decoder and our proposed decoders achieve lower DFR levels than the standard decoding techniques currently employed in QC-MDPC based cryptosystems. In particular, including a number of BF-Max iterations before switching to the parallel decoding strategy drastically improves the correction capability w.r.t. the standard parallel decoder. Moreover, by executing a small number of BF-Max iterations (i.e., $0.3t$ iterations), we are able to achieve a lower DFR than the BGF decoder. Finally, in Figure 4 we can see the benefits of the *restart* strategy by comparing the DFR of BFH$(0.4t, 3)$ and BFH-R$(0.4t, 3)$: the DFR of the second algorithm is lower by two orders of magnitude, also improving on the plain BF-Max decoder while executing essentially the same number of operations as the BFH$(0.4t, 3)$ decoder.

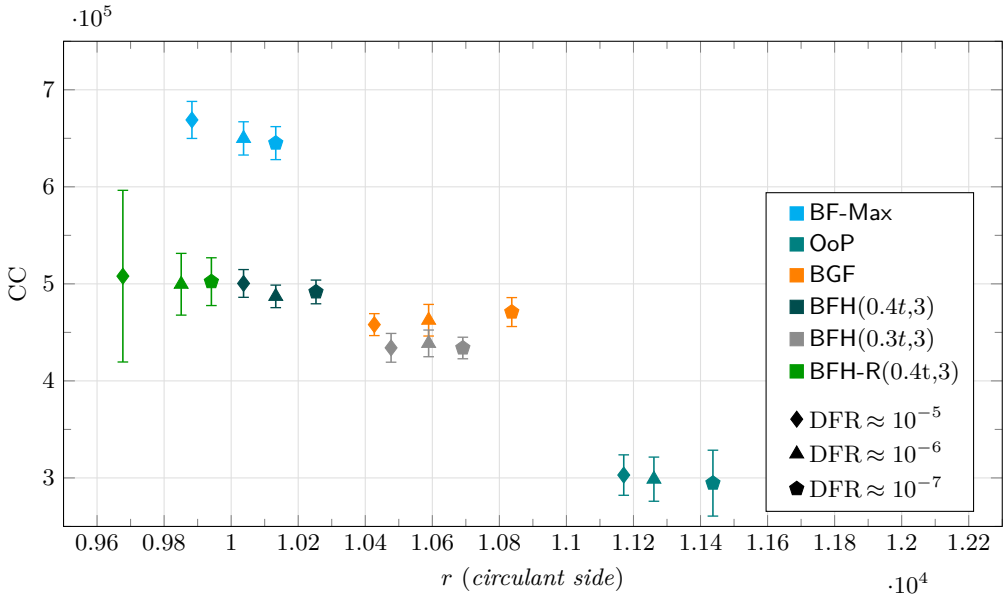


Fig. 5. Circulant block size (r) vs clock cycles (and standard deviations) for various DFR levels and decoders.

We now investigate the performance of a BIKE-like IND-CPA KEM instantiated with our family of decoders. Figure 5 shows, for all the analyzed decoders, the size of the circulant block r and the clock cycle count required by each decoder to achieve different DFR levels in $\{10^{-5}, 10^{-6}, 10^{-7}\}$. Starting from BF-Max, we have that the smallest value of r resulting from our experimental campaign (9883) is 19.8% smaller than the value employed in BIKE (12323) and 5.8% smaller than the value employed in LeanBIKE (10499). Among our proposed hybrid decoders, BFH-R$(0.4t, 3)$ achieves a circulant block of size $r = 9677$, which is 21.5% smaller than the value employed in BIKE and 7.8% smaller than the one in LeanBIKE.

Table 4. Public key size, ciphertext size, and number of clock cycles required to complete an ephemeral KEM with different DFR targets ($10^{-5}, 10^{-6}, 10^{-7}$) and decoders (BGF, OoP, BF-Max, BFH-R($0.4t, 3$)).

DFR	**Decoder**	r (bits)	$\|\mathbf{pk}\| = \|\mathbf{ct}\|$ (B)	**Keygen** (kCC)	**Enc** (kCC)	**Dec** (kCC)	$\|\mathbf{pk}\| + \|\mathbf{ct}\|$ (B)	**KEM** (kCC)
10^{-5}	BGF	10 427	1 304	386	86	428	2 608	900
	OoP	11 171	1 397	402	88	281	2 794	771
	BF-Max	9 883	1 236	378	86	622	2 472	1 086
	BFH-R	9 677	1 210	391	85	464	2 420	940
10^{-6}	BGF	10 589	1 324	396	87	432	2 648	915
	OoP	11 261	1 408	476	88	278	2 816	842
	BF-Max	10 037	1 255	399	86	626	2 510	1 111
	BFH-R	9 851	1 232	402	86	480	2 464	968
10^{-7}	BGF	10 837	1 355	383	87	439	2 710	909
	OoP	11 437	1 430	409	88	284	2 860	781
	BF-Max	10 133	1 267	397	86	625	2 534	1 108
	BFH-R	9 941	1 243	395	86	473	2 486	954

Such decoder exhibits a higher variance in execution time than the others due to the restart mechanism, that does not however alter the performance of the decoder significantly. For DFR $\leq 10^{-6}$, the BGF decoder is Pareto-dominated by the BFH($0.3t, 3$) decoder, which requires 5% fewer clock cycles while employing the same circulant block size.

As a final comparison, we test the set of decoders designed and implemented in this work as a drop-in replacement for the BGF decoder in LeanBIKE, and evaluate the performance of the resulting ephemeral KEM. In Table 4, we report the size of the public key and ciphertext along with the number of clock cycles required to execute the Keygen, Enc and Dec primitives, for each DFR level and decoder choice.

We set the circulant block size r of each instance as the smallest value guaranteeing a low enough DFR. We also report the total amount of bytes to be exchanged, and the clock cycle count of the complete KEM. As can be seen, by employing the set of decoders described and optimized in the previous sections (in particular, BF-Max and BFH-R($0.4t, 3$)) we are able to obtain a more compact KEM. We achieve an overall bandwidth requirement reduction of up to 694 bytes with respect to BIKE (whose figures are reported in Table 3) and 224 bytes with respect to the three-iteration BGF decoder (for DFR $\approx 10^{-7}$), with an increase in clock cycle count of at most 53 kCC. When comparing against the speed-optimized OoP decoder, we reduce the combined size of the public key and ciphertext by up to 374 bytes. Our smallest public key and ciphertext size (1210 bytes) allows the resulting QC-MDPC based KEM to be employed in several UDP-based protocols such as IKEv2, DTLS 1.3, and QUIC, for all DFR levels and with at least 91 bytes remaining for additional data even when considering a hybrid key exchange (i.e., when including the classical KEX payload).

Quantifying timing information leakage. The ephemeral use of a key-pair in our proposed approach provides a natural stopgap against information leakage via timing side channel attacks; indeed, an attacker has a single timing sample related to a specific key-pair-encapsulated secret pair. While this is often accepted in practice, we provide the results of two numerical experiments

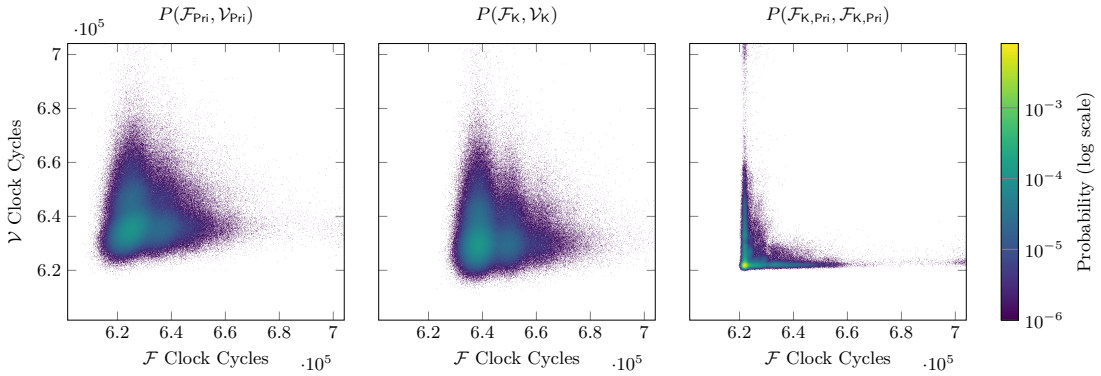


Fig. 6. Joint probability distribution $P(\mathcal{F}, \mathcal{V})$ estimated for the three timing leakage experiments.

to quantify the amount of information in the hands of an attacker. The two scenarios we model are those of an attacker willing to find out whether the encapsulated secret K has a specific value, or whether the private key Pri has a specific value. The first experiment measures the joint distribution of the two random variables $\mathcal{F}_{\mathsf{K}}$ and $\mathcal{V}_{\mathsf{K}}$, which model the execution time of the decapsulation of K under a fixed keypair, and under a uniformly randomly drawn keypair. The joint distribution is estimated by measuring the time taken to decapsulate the same shared secret K (which is operatively implemented by employing the same value of the seed in the encapsulation function) under either the same, fixed private key, or a randomly drawn one. Note that, in the second case, the value of Pub employed in the encapsulation is the appropriate one (i.e., the public key corresponding to the randomly drawn Pri), so that the corresponding decapsulation should succeed, bar decoding failures. The second experiment shares the same structure, while measuring the joint distribution of $\mathcal{F}_{\mathsf{Pri}}$ and $\mathcal{V}_{\mathsf{Pri}}$, which model the execution time of the decapsulation of either a fixed K, or a uniformly randomly drawn encapsulated secret K. In both cases, the pairs of samples forming a sample of the joint distribution are linked together by being obtained with the same value of K (for the first experiment) or being decapsulated under the same Pri (for the second experiment).

We measure the clock cycles taken for a $\mathsf{BF\text{-}Max}$ decapsulation, across 10^6 samples, with the same experimental setup described in the previous section. The values are obtained via the `rdtscp` instruction and grouped into 128-clock cycles-wide histogram bins (corresponding to 53 ns at our running frequency), given the moderately large range of values (the average standard deviation is 21.69 kCC). Once the sample joint distributions $(\mathcal{F}_{\mathsf{Pri}}, \mathcal{V}_{\mathsf{Pri}})$ and $(\mathcal{F}_{\mathsf{K}}, \mathcal{V}_{\mathsf{K}})$ are available, we compute the sample mutual information values $I(\mathcal{F}_{\mathsf{Pri}}; \mathcal{V}_{\mathsf{Pri}})$ and $I(\mathcal{F}_{\mathsf{K}}; \mathcal{V}_{\mathsf{K}})$ reported in Table 5. Comparing the value of the mutual information $I(\mathcal{F}_{\mathsf{Pri}}; \mathcal{V}_{\mathsf{Pri}})$ with the entropies $H(\mathcal{F}_{\mathsf{Pri}})$ and $H(\mathcal{V}_{\mathsf{Pri}})$ of the involved random variables provides a way to quantify how much a single timing sample allows an attacker to distinguish a decapsulation running with a specific Pri from one running with a random one (the same line of reasoning applies to $(\mathcal{F}_{\mathsf{K}}, \mathcal{V}_{\mathsf{K}})$). Finally, as a touchstone of the information content of a sample of the clock cycle counter stemming from the benchmarking platform itself, we run a last experiment which simply repeats twice the same key generation, encapsulation and decapsulation computation. This experiment effectively estimates the noise floor as the sample distribution of $(\mathcal{F}_{\mathsf{K,Pri}}, \mathcal{F}_{\mathsf{K,Pri}})$, where $\mathcal{F}_{\mathsf{K,Pri}}$ models the value of the clock cycle counter of an execution of the KEM primitives while keeping the value of both the encapsulated key and the keypair fixed.

Table 5. Entropy H and mutual information I estimated for the three timing leakage experiments.

Estimated distribution	$H(\mathcal{F})$	$H(\mathcal{V})$	$I(\mathcal{F};\mathcal{V})$
$(\mathcal{F}_{\mathsf{Pri}}, \mathcal{V}_{\mathsf{Pri}})$	7.696	7.740	0.168
$(\mathcal{F}_{\mathsf{K}}, \mathcal{V}_{\mathsf{K}})$	7.542	7.761	0.136
$(\mathcal{F}_{\mathsf{K,Pri}}, \mathcal{F}_{\mathsf{K,Pri}})$	5.341	5.332	0.064

Figure 6 depicts the obtained sample joint probability distributions employing as axes the ranges of the marginal p.m.f.s (the distributions are discrete), while Table 5 reports the values of the entropy of all marginal distributions, and the mutual information values obtained from each joint one. A first observation is the fact that an extremely conservative upper bound about the information that an attacker can obtain from a single timing measure with a quantization of 128 CCs (53ns at our running frequency) is $\max(H(\mathcal{V}_{\mathsf{Pri}}), H(\mathcal{V}_{\mathsf{K}}))$. This very conservative bound assumes that the entire information contained in the clock cycle counter is indeed related to the secret material, and exploitable. A first refinement of the estimate can be done observing that the information contained in one clock cycle counter sample when an execution with fixed Pri and K is run, i.e., $H(\mathcal{F}_{\mathsf{K,Pri}})$ is not related to either fixed quantities, and thus provides a good proxy of the information content related to the running environment (e.g., due to OS, microarchitectural and frequency jitter). As reported in Table 5 $H(\mathcal{F}_{\mathsf{K,Pri}})$ is $\approx$ 5.3 bits of entropy, representing the majority of the $\max(H(\mathcal{V}_{\mathsf{Pri}}), H(\mathcal{V}_{\mathsf{K}})) = 7.76$b of the aforementioned conservative estimate. Therefore, a more precise information leakage estimate points to 2.4 bits of information coming from input-dependent behavior. Finally, we consider the obtained values of the mutual information in our experiments, and observe that the information an attacker can extract from an unknown K and Pri measure is the maximum value of I from Table 5, which is 0.168 bits. Moreover, we note that computing the mutual information between two all-values-fixed samples is 0.064 bits (likely due to the system level interactions of two consecutive runs), leaving the excess of mutual information of the actual experiments upper bounded by 0.104 bits. Summarizing our experiments, we can therefore state that even considering a very conservative estimate of the information gained by the attacker, and assuming perfect exploitation of this information, the security margin is reduced by at most 2.4 bits, with a more reasonable estimate being a 0.104 bit reduction, which we deem negligible.

6 Conclusion

In this work, we presented an AVX2-optimized implementation of the BF-Max decoder[5], and proposed a new family of hybrid decoders for use in code-based KEMs targeted to ephemeral-use scenarios. Our proposed decoder strategies allow a designer to choose a trade-off between error correction capability (which influences public key and ciphertext size) and execution time. These design and implementation techniques improve the Decoding Failure Rate of state-of-the-art decoders at a minimal computational overhead, achieving a 7.8% bandwidth improvement over LeanBIKE. This reduction enables the use of code-based ephemeral KEMs in constrained UDP-based protocols such as IKEv2, while still leaving additional room for pre-quantum key exchange and optional data.

[5] The source code will be made available before publication.

References

[1] G. Alagic et al. *Status report on the fourth round of the NIST post-quantum cryptography standardization process*. U.S. Department of Commerce, National Institute of Standards and Technology - NIST Internal Report - NIST IR 8545. URL: https://doi.org/10.6028/NIST.IR.8545. 2025.

[2] N. Aragon et al. *BIKE: Bit Flipping Key Encapsulation – Algorithm Specification and Supporting Documentation (Round 4)*. U.S. Department of Commerce, National Institute of Standards and Technology - PQC Standardization Process - Round 4 Submissions: Public-key Encryption and Key-establishment Algorithms. https://bikesuite.org. 2022.

[3] A. Baldelli, M. Baldi, F. Chiaraluce, and P. Santini. "BF-Max: an Efficient Bit Flipping Decoder with Predictable Decoding Failure Rate". In: *2025 IEEE International Symposium on Information Theory (ISIT)*. 2025, pp. 1–6. DOI: 10.1109/ISIT63088.2025.11195380.

[4] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. *LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems*. U.S. Department of Commerce, National Institute of Standards and Technology - PQC Standardization Process - Round 2 Submissions: Public-key Encryption and Key-establishment Algorithms. URL: https://www.ledacrypt.org/. 2020.

[5] D. J. Bernstein et al. "Classic McEliece: Conservative Code-Based Cryptography". In: *NIST Post-Quantum Cryptography Standardization – Round 1 Submission*. 2017.

[6] N. Drucker, S. Gueron, and D. Kostic. *A lean BIKE KEM design for ephemeral key agreement*. presented at the sixth NIST Post-Quantum Cryptography Standardization Conference, extended abstract available at https://www.amazon.science/publications/a-lean-bike-kem-design-for-ephemeral-key-agreement. 2024.

[7] N. Drucker, S. Gueron, and D. Kostic. "QC-MDPC Decoders with Several Shades of Gray". In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*. Ed. by J. Ding and J. Tillich. Vol. 12100. Lecture Notes in Computer Science. Springer, 2020, pp. 35–50. DOI: 10.1007/978-3-030-44223-1_3.

[8] R. G. Gallager. "Low-density parity-check codes". In: *IRE Trans. Inf. Theory* 8.1 (1962), pp. 21–28. DOI: 10.1109/TIT.1962.1057683. URL: https://doi.org/10.1109/TIT.1962.1057683.

[9] Q. Guo, T. Johansson, and P. Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". In: *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 789–815. DOI: 10.1007/978-3-662-53887-6_29.

[10] Q. Guo, T. Johansson, and P. S. Wagner. "A Key Recovery Reaction Attack on QC-MDPC". In: *IEEE Trans. Inf. Theory* 65.3 (2019), pp. 1845–1861. DOI: 10.1109/TIT.2018.2877458.

[11] D. Hofheinz, K. Hövelmanns, and E. Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *Theory of Cryptography – TCC 2017*. Vol. 10677. Lecture Notes in Computer Science. Springer, 2017, pp. 341–371. DOI: 10.1007/978-3-319-70500-2_12.

[12] L. Huguenin-Dumittan and S. Vaudenay. "On IND-qCCA Security in the ROM and Its Applications - CPA Security Is Sufficient for TLS 1.3". In: *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III*. Ed. by O.

Dunkelman and S. Dziembowski. Vol. 13277. Lecture Notes in Computer Science. Springer, 2022, pp. 613–642. DOI: 10.1007/978-3-031-07082-2_22.

[13] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000.

[14] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 7296. Oct. 2014. DOI: 10.17487/RFC7296.

[15] C. A. Melchor et al. *HQC: Hamming Quasi-Cyclic – Algorithm Specification and Supporting Documentation (Round 4)*. U.S. Department of Commerce, National Institute of Standards and Technology - PQC Standardization Process - Round 4 Submissions: Public-key Encryption and Key-establishment Algorithms. URL: https://pqc-hqc.org. 2022.

[16] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto. "MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes". In: *2013 IEEE International Symposium on Information Theory (ISIT)*. 2013, pp. 2069–2073.

[17] E. Prange. "The use of information sets in decoding cyclic codes". In: *IRE Transactions on Information Theory* 8.5 (1962), pp. 5–9. DOI: 10.1109/TIT.1962.1057777.

[18] E. Rescorla, H. Tschofenig, and N. Modadugu. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*. RFC 9147. Apr. 2022. DOI: 10.17487/RFC9147.

[19] P. Santini, D. De Zuane, A. Baldelli, and M. Baldi. *Flip a Failure into a Success: Improved Bit Flipping Decoding for QC-MDPC Codes*. Cryptology ePrint Archive, Paper 2026/1616. 2026. URL: https://eprint.iacr.org/2026/1616.

[20] P. Schwabe, D. Stebila, and T. Wiggers. "Post-Quantum TLS Without Handshake Signatures". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by J. Ligatti, X. Ou, J. Katz, and G. Vigna. ACM, 2020, pp. 1461–1480. DOI: 10.1145/3372297.3423350.

[21] N. Sendrier. "Decoding One Out of Many". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*. Ed. by B. Yang. Vol. 7071. Lecture Notes in Computer Science. Springer, 2011, pp. 51–67. DOI: 10.1007/978-3-642-25405-5_4.

[22] J. Stern. "A method for finding codewords of small weight". In: *Coding Theory and Applications*. Ed. by G. D. Cohen and J. Wolfmann. Vol. 388. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 1989, pp. 106–113. DOI: 10.1007/BFb0019850.

[23] T. Wang, A. Wang, and X. Wang. "Exploring Decryption Failures of BIKE: New Class of Weak Keys and Key Recovery Attacks". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 70–100. DOI: 10.1007/978-3-031-38548-3_3.

[24] B. Zhou, H. Jiang, and Y. Zhao. "CPA-Secure KEMs are also Sufficient for Post-quantum TLS 1.3". In: *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part III*. Ed. by K. Chung and Y. Sasaki. Vol. 15486. Lecture Notes in Computer Science. Springer, 2024, pp. 433–464. DOI: 10.1007/978-981-96-0891-1_14.